unit I

Introduction to computer Graphics

chap 1

1) Architecture of vector display system
2) Architecture of Raster display system
3) Application of computer graphics
4) Difference between vector and Raster display.

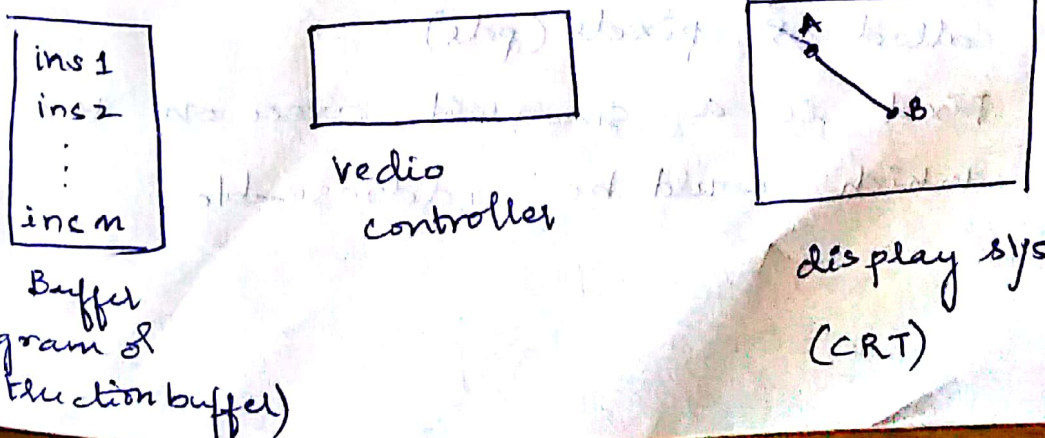
Introduction

- Area of computer graphics initially came into existence with the display of textual data over printing plotters and CRT monitors
- Later it evolved to display, store and manipulate different images and models of real time objects
- Todays computer graphics has mainly emerged to be interactive where the area itself is called as interactive graphics - user interacting with computer system using graphical notation

eg: OS being GUI

Vector display System

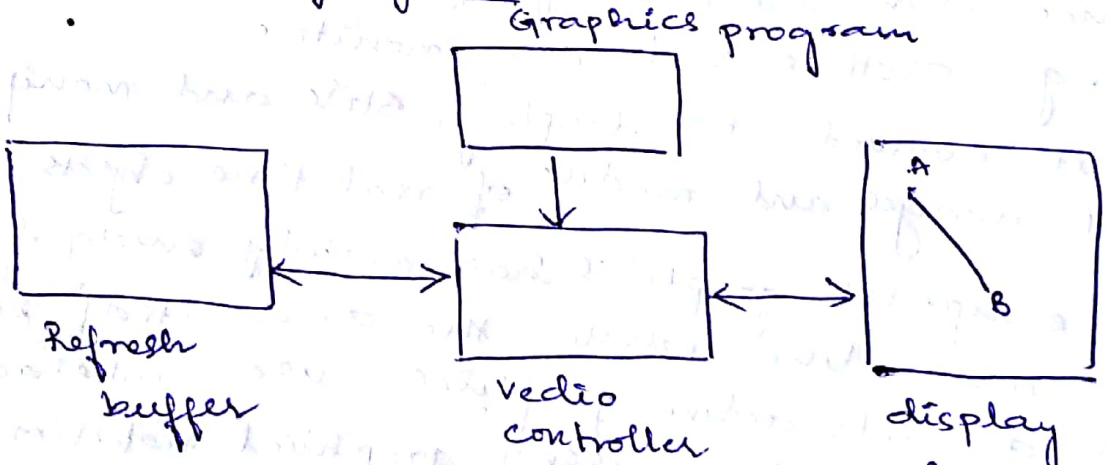
Early in 1980's computer graphics used vector display system in the CRT monitors to display the information.



CRT has a phospha plate.
It has a disadvantage
After few time phospha plate decays.

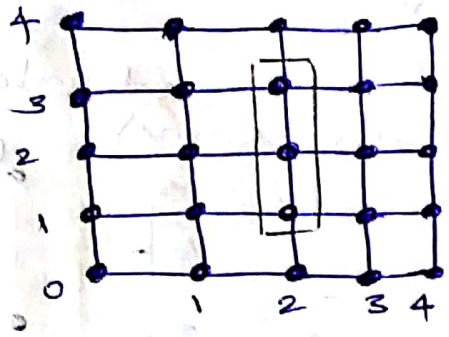
- programing is difficult
- If program length exceeds there will be flickering effect in display system

Raster display system



It uses concept of Bitmap Graphics System
In Bitmap Graphics System whatever is to be displayed is stored in binary values of 1
The entire system is represented as a grid of closely held points and each point is called as pixels (pels)
Pixel is a smallest area on the screen which could be addressable.

eg: 5x5 display system



to glow the line the program code is

```
0 0 0 0 0
0 0 1 0 0
0 0 1 0 0
0 0 1 0 0
0 0 0 0 0
```

⇒ Raster scan process

Chap 2 **Raster Graphics Algorithm**

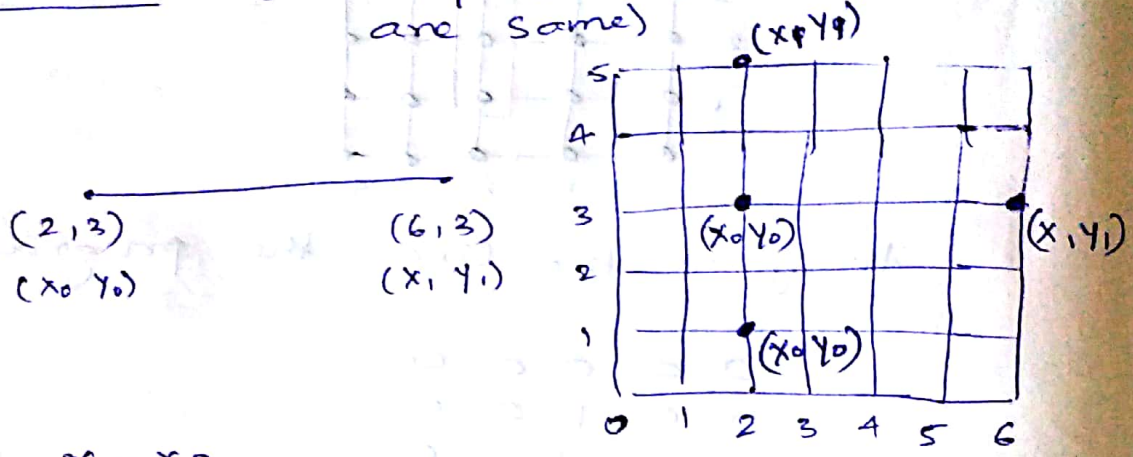
Scan Converting lines

concepts on line

- To mark a line we need 2 points
  1) starting point → $(X_0\ Y_0)$ } pixel.
  2) ending point → $(X_1\ Y_1)$ }
- Always $X_0 <= X_1$ & $Y_0 <= Y_1$
- First pixel and final pixel is the fixed pixel for plotting.

Simple line plotting Algorithm

1: Horizontal line (when y coordinates of both points are same)

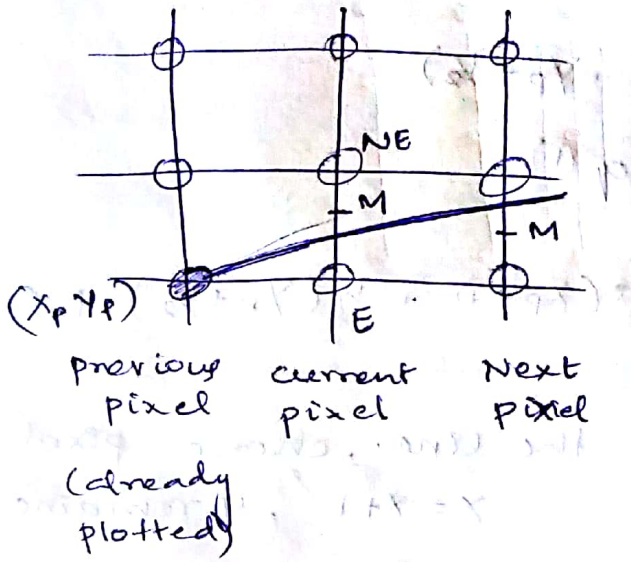



```
x = x0
y = y0
plot (x, y);
    while (x < x1)
      {
        x = x + 1;
        plot (x, y);
      }
```

2. Vertical line (when x coordinates of both points are same)


```
X = X0
Y = Y0
Plot (X, Y)
    while (Y < Y1)
      {
        Y = Y + 1;
        Plot (X, Y);
      }
```

Mid point line Algorithm:



Here x always changes but y depends on the decision, it may or may not change

The line equation for a point (x, y) lying on the line is given by

$$F(x,y) = ax + by + c = 0 \longrightarrow (1)$$

Slope intercept equation for y is given as.

$$y = x \cdot \frac{dy}{dx} + B \longrightarrow (2)$$

Here $dy = y_1 - y_0$, $dx = x_1 - x_0$

Simplifying (2)

$$y \cdot dx = x \cdot dy + B\,dx.$$

Rearranging the equation to the form of equation (1)

$$dy \cdot x - y\,dx + B\,dx$$

where $a = dy$, $b = -dx$, $c = B\,dx$

Hence

$$F(x,y) = ax + by + c$$

Selecting the current pixel

$$d = F(M) = F(x_p + 1, y_p + \tfrac{1}{2})$$

Substitute in eq (1)

$$F(x_p + 1, y_p + \tfrac{1}{2}) = a(x_p + 1) + b(y_p + \tfrac{1}{2}) + c$$

If, $d = 0$, M is on the line., choose pixel E,
$x = x + 1$, y remains same

If, $d < 0$, M is above the line,
choose pixel E

$$x = x + 1$$
$$y = y$$

If, $d > 0$, M is below the line
choose pixel NE,

$$x = x + 1$$
$$y = y + 1$$

To select the next pixel:

If E is selected,

$$d_{new} = F(x_p + 2, y_p + \tfrac{1}{2})$$

$$F(x_p + 2, y_p + \tfrac{1}{2})$$

$$= a(x_p + 2) + b(y_p + \tfrac{1}{2}) + c$$

we know that $d_{old} = a(x_p + 1) + b(y_p + \tfrac{1}{2}) + c$

$$d_{new} - d_{old} = a(x_p + 2) + b(y_p + \tfrac{1}{2}) + c - [a(x_p + 1) + b(y_p + \tfrac{1}{2}) + c]$$

$= a(x_p+2) - a(x_p+1)$

$= ax_p + 2a - ax_p + a = a$

$d_{new} = d_{old} + a$

$= d_{old} + dy$ $(\because dy = incrE)$

$$\boxed{d_{new} = d_{old} + incrE}$$

If NE is selected,

$d_{new} = F(x_p+2, y_p+3/2)$

$d_{new} = F(x_p+2, y_p+3/2)$

$= a(x_p+2) + b(y_p+3/2) + c$

we know that,

$d_{old} = a(x_p+1) + b(y_p+1/2) + c$

$d_{new} - d_{old} = a(x_p+2) + b(y_p+3/2) + c$

$- [a(x_p+1) + b(y_p+1/2) + c]$

$= ax_p + 2a + by_p + \frac{3b}{2} + c - ax_p$

$- a - by_p - b/2 - c$

$d_{new} - d_{old} = a + b$

$d_{new} = d_{old} + a + b$

$= d_{old} + dy - dx.$ $\underbrace{dy - dx}_{\text{incrNE}}$

$$\boxed{d_{new} = d_{old} + incrNE}$$

To get the first decision

$$d_{init} = F(x_0 + 1, y_0 + 1/2)$$

$$= a(x_0 + 1) + b(y_0 + 1/2) + c$$

$$= ax_0 + a + by_0 + b/2 + c$$

$$= 0 + b/2 + c$$

$$= ax_0 + by_0 + c + a + b/2$$

$$= 0 + a + b/2$$

$$= a + b/2$$

$$= dy - dx/2$$

$$\boxed{2d_{init} = 2 * dy - dx}$$

ALGORITHM

step1: [Read inputs]

Read x_0, y_0

Read x_1, y_1

step2: calculate dx and dy

$dx = x_1 - x_0$

$dy = y_1 - y_0$

step3: calculate incrE and incrNE

incrE = dy

incrNE = dy - dx

step4 : [calculate initial d]

$d = 2*dy - dx$

step5 : [Assign x and y]

$x = x_0$

$y = y_0$

step6 : [Plot first point]

plot (x, y)

step7 : while (x < xD)

```
{
  if (d<=0)
  {
    x = x+1;
    d = d + 2xincrE
  }
  else
  {
    x = x+1
    y = y+1
    d = d + 2* incrNE
  }
  plot (x,y)
}
```

step8 : stop

Problem :

1] using midpoint line algorithm plot pixel for the line (5, 8) and (9, 11)

x_0 y_0 x_1 y_1

→ $x_0 = 5$ $y_0 = 8$

$x_1 = 9$ $y_1 = 11$

$dx = x_1 - x_0 = 9 - 5 = 4$

$dy = y_1 - y_0 = 11 - 8 = 3$

incrE = $dy = 3$

incrNE = $dy - dx = 3 - 4 = -1$

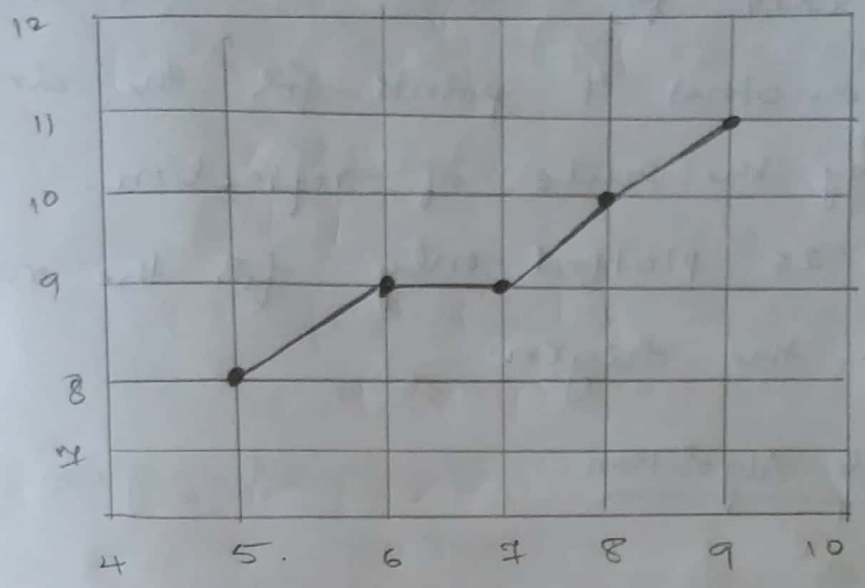
$d = 2 * dy - dx = 6 - 4 = 2$

$x = x_0 = 5$

$Y = y_0 = 8$

plot (5, 8)

dx=0?	d>0?	d	x	y	plot(x,y)
–	Yes	d = d + 2x incrNE = 0	6	9	plot (6,9)
Yes	–	d = d + 2* incrE = 0 + 6 = 6	7	9	plot (7,9)
–	Yes	d = d + 2*incrNE = 6 - 2 = 4	8	10	plot (8,10)
–	Yes	d = d + 2*incrNE = 4 - 2 = 2	9	11	plot (9,11)



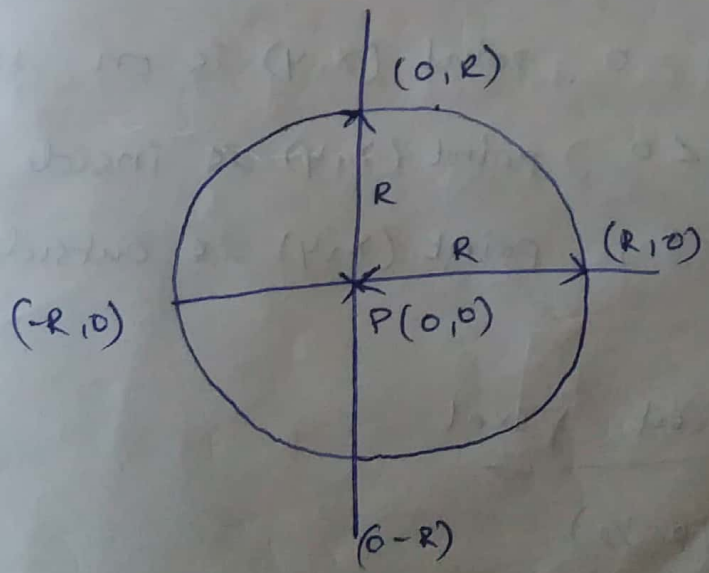
Scan converting Circles

* Midpoint circle Algorithm

circle centre is at origin (0,0)

input: Radius → R value

centre → P(x, y)



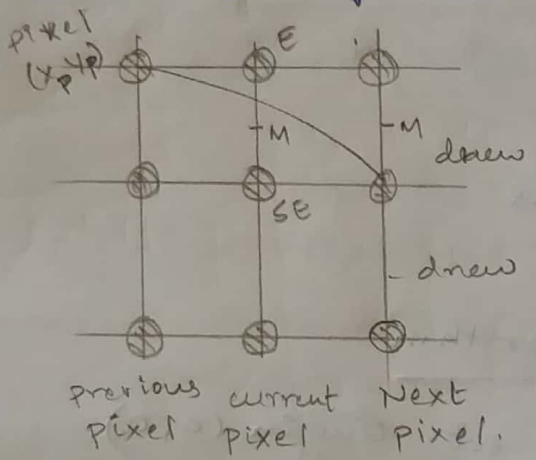
Eight way symmetry

If a point (x,y) for a circle to be plotted is derived then another 7 points for the circle could be derived by the rule of reflection.

A point is plotted only for the radius of 45° from the origin.

Midpoint circle Algorithm

SE is given priority



Equation of the circle for every point (x,y) is given by

$$F(x,y) = x^2 + y^2 - R^2 \quad \text{—— (1)}$$

If $F(x,y) = 0$, point (x,y) is on the circle.

If $F(x,y) < 0$, point (x,y) is inside the circle

If $F(x,y) \geq 0$, point (x,y) is outside the circle.

To select current pixel

$$M = (x_p + 1, y_p - 1/2)$$

applying M over equation (1) to get d

$d = F(x_{p+1}, y_{p-1/2}) = (x_p+1)^2 + (y_p - 1/2)^2 - R^2$

if $(d=0)$, point is on the circle, choose SE

$x = x+1$

$y = y-1$

if $d<0$, point is inside the circle, choose E, (boundary is closer to E)

$x = x+1$

if $d>0$, point is outside the circle, choose SE

$x = x+1$

$y = y-1$

To select the next pixel

If E is selected

$M = (x_p+2, y_p - 1/2)$

Apply M to equation 1

$d_{new} = F(x_p+2, y_p - 1/2)$

$= (x_p+2)^2 + (y_p - 1/2)^2 - R^2.$

we know that

$d_{old} = (x_p+1)^2 + (y_p - 1/2)^2 - R^2$

$d_{new} - d_{old} = [(x_p+2)^2 + (y_p - 1/2)^2 - R^2] - [(x_p+1)^2 + (y_p - 1/2)^2 - R^2]$

$d_{new} - d_{old} = x_p^2 + 4x_p + 4 - x_p^2 - 2x_p - 1$

$d_{new} - d_{old} = 2x_p + 3$

$$\boxed{d_{new} = d_{old} + 2x_p + 3}$$

If SE is selected

$$M = (x_p + 2 \quad y_p + 3/2)$$

applying M to equation (1)

$$d_{new} = F(x_p + 2, y_p + 3/2)$$

$$= (x_p + 2)^2 + (y_p + 3/2)^2 - R^2$$

We know that

$$d_{old} = F(x_p + 1, y_p - 1/2) = (x_p + 1)^2 + (y_p - 1/2)^2 - R^2.$$

$$d_{new} - d_{old} = [(x_p + 2)^2 + (y_p + 3/2)^2 - R^2] - [(x_p + 1)^2 + (y_p - 1/2)^2 - R^2]$$

$$= x_p^2 + 4x_p + 4 + y_p^2 + 3y_p + 9/4 - x_p^2 - 1 - 2x_p - y_p^2 - 1/4 + y_p$$

$$\boxed{d_{new} - d_{old} = 2x_p - 2y_p + 5}$$

To get the first (initial decision)

$$d_{init} = F(1, R - 1/2), \text{ apply to equation (1)}$$

$$= 1^2 + (R - 1/2)^2 - R^2$$

$$= 1 + R^2 + 1/4 - R - R^2$$

$$\boxed{d_{init} = 5/4 - R}$$

Algorithm for Midpoint Circle

Step1 : [Read the inputs]

Read R

Step2 : [calculate dinit value]

$d = 5.0/4.0 - R$

Step3 : [Initialise x and y value]

```
x=0;
Y=R;
```

[Plot the ential point]

```
plot (x,y)
plot (-x ,y)
plot (x, -y)
plot (-x ,-y)
plot (y, x)
plot (-y ,x)
plot (y, -x)
plot (-y, -x)
```

Step5 : While (Y > x)

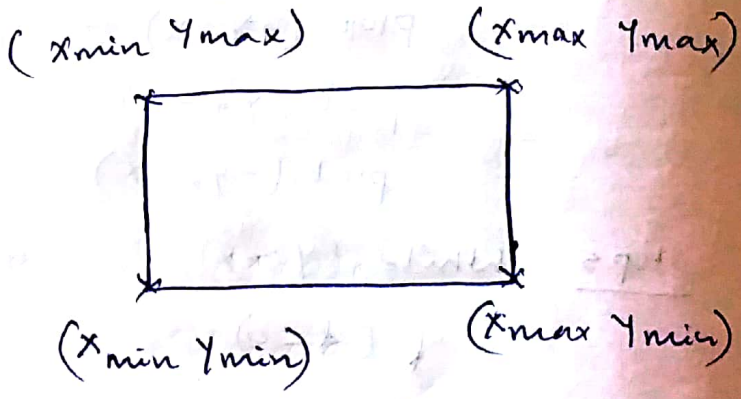
```
if (d<0)
 {
   x=x+1
   d=d + 2x + 3
 }
else
 {
   x = x+1 ;
   y = y-1;
   d = d + 2(x-y) +5
 }
```

plot (x y)
plot (-x, y)
plot (x, -y)
plot (-x, -y)
plot (y, x)
plot (-y, x)
plot (y, -x)
plot (-y, -x)

Step 6: [END]

Filling Rectangles

Filling rectangle is the mechanism of coloring the pixels of the rectangle those which lie inside the boundary of the rectangle.



Algorithm

Step 1: [Read Inputs]

Read xmin, ymin
Read xmax ymin
Read xmin, ymax
Read xmax, ymax

Step 2 :
```
for (Y = Ymin ; Y <= Ymax ; Y++)
    {
    for (X = Xmin ; X <= Xmax ; X++)
        {
        Plot (X, Y, color);
        }
    }
```

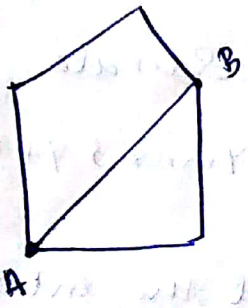
Step 3 : [END]

imp *** Filling Polygons -(imp *** problems)

Two types · 1) convex polygon
2) concave polygon

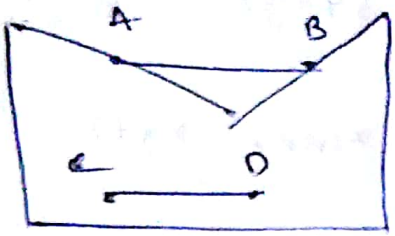
1) convex polygon

A convex polygon is one in which the points connecting the line segments ~~to~~ is inside the polygon, then the entire line segment will be inside the polygon.



2) Concave polygon

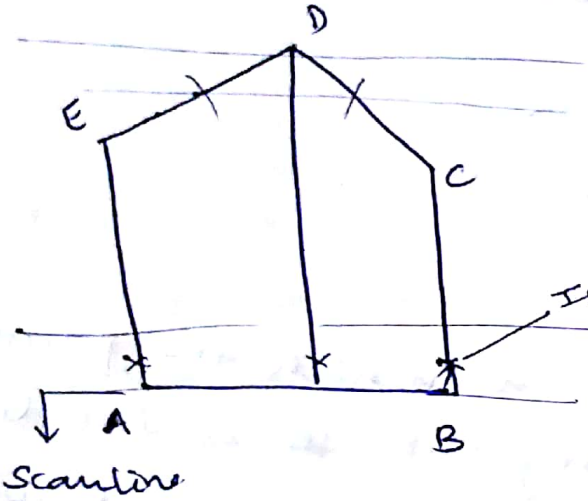
A concave polygon is the one in which ~~(the line segment connecting the)~~ if the points connecting the line segment is inside the polygon then, the line segment may or maynot be inside the polygon.



Filling of convex Polygon

Scan line and edge-detection Algorithm

[horizontal line that starts with smallest y coordinate value and ends at highest y coordinate value]



lowest x value - left edge

highest x value - right edge.

Algorithm

Ymin = Smallest y coordinate

Ymax = largest y coordinate

```
for (Y = Ymin ; Y <= Ymax ; Y++)
  {
  array a[] = find the intersection points.
(le) leftedge = Smallest value of a[]
(re) rightedge = largest value of a[]
    plot (le, re, color)
}.
```

Edge Coherence and Scan line Algorithm

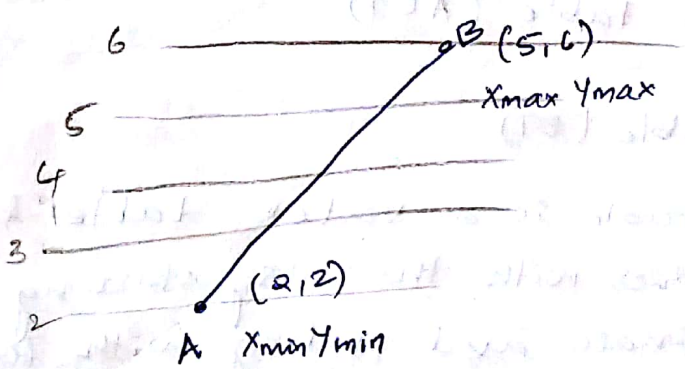
This algorithm is used to fill both convex and concave polygon in a efficient manner.

It uses two different concepts.

1) Edge coherence
2) Scanline method.

The algorithm uses scan line between the end points of the edges to access all the vertical points corresponding to an edge and in each edge to find the span of the edge the algorithm uses the concept called as edge coherence.

So Basically edge coherence is a property that is used to get the x intersection with respect to a particular scan line



$$x_2 = x_i = 2$$

$$x_{i+1} = x_i + 1/m$$

$$m = \frac{y_{max} - y_{min}}{x_{max} - x_{min}}$$

$$= 4/3$$

$$\therefore \boxed{1/m = 3/4}$$

$$x_3 = x_2 + 3/4 = 2 + 3/4 = 2.75$$

Edge coherence is a property which states that if a scan line $i+1$ comes in between the end points of an edge then the intersection x_{i+1} is equal to $x_i + 1/m$

ie $x_{i+1} = x_i + \frac{1}{m}$ where x_i is the x intersection value w.r.t previous scan line i and m is the slope of an edge lying between the points $(x_{min}\ y_{min})$ & $(x_{max}\ y_{max})$

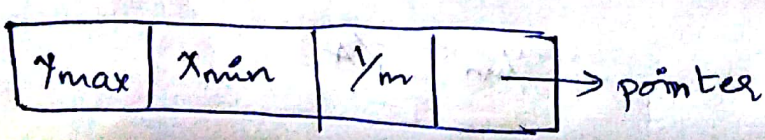
where $m = \frac{y_{max} - y_{min}}{x_{max} - x_{min}}$.

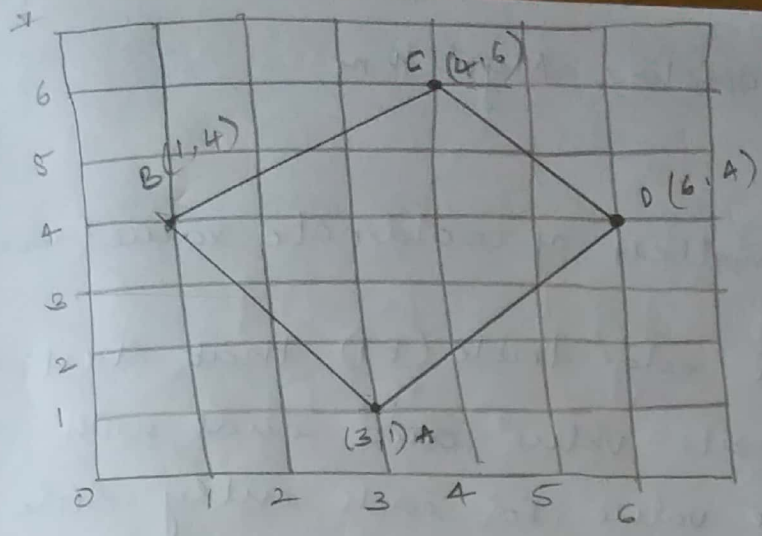
Edge coherence and scanline algorithm uses two data structures

1) Global Edgetable (ET)
2) Active Edge table (AET)

Global edge table (ET)

- Global edge table is a vertex table having n number of entries with the entry starting from lowest y coordinate and ending with the highest y coordinate value.
- Each entry in the ET stores the information of an edge in the increasing order of x value and is called as bucket of the entry
- Each bucket is a node representing four fields and is given by





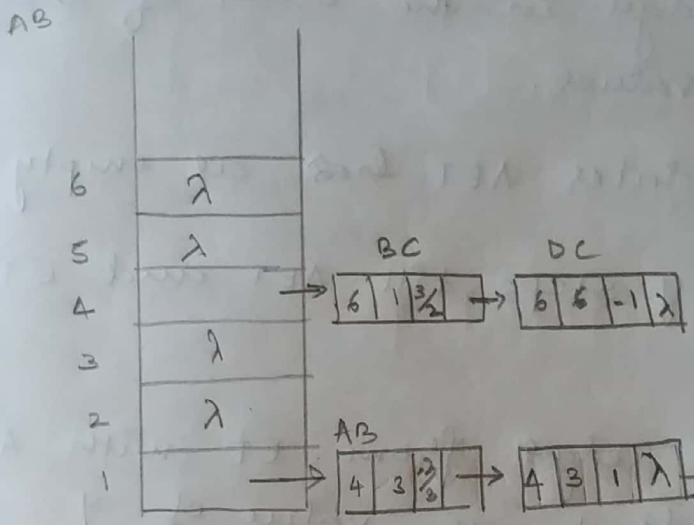
A (3,1)
B (1,4)
C (4,6)
D (6,4)

A (3,1) B (1,4)
x_{min} y_{min} x_{max} y_{max}

$m = \frac{4-1}{1-3} = \frac{3}{-2}$

D (6,4) C (4,6)
x_{min} y_{min}

$\frac{6-4}{4-6} = \frac{2}{-2} = -1$



A (3,1) D (6,4)
x_{min} y_{min} x_{max} y_{max}

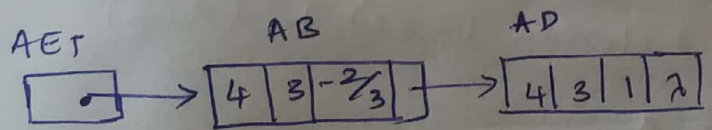
B (1,4) C (4,6)
(x_{min} y_{mix}) (x_{max} y_{max})

$m \frac{6-4}{4-1} = \frac{2}{3}$.

Active Edge Table (AET)

Active edge table is a data structure that holds all the active edges corresponding to a particular scan line.

Eg: For scan line $y = 1$



Imp/ Edge coherence and scanline Algorithm

Algo:

Step 1 : Initialise y = smallest y coordinate value (y_{min})

Step 2 : Create a Global edge table (ET) that starts from smallest y coordinate value and ends with largest y coordinate value. For each entry store information of edges in the order of their increasing x_{min} values.

Step 3: Initialise the pointer AET ~~list~~ as empty (λ)

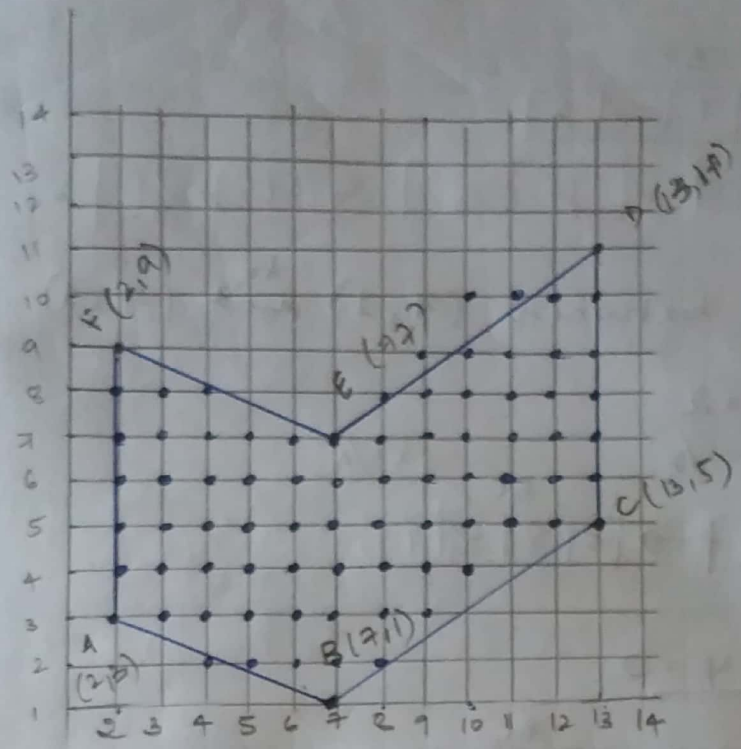
Step 4: Do the following steps till AET and ET are empty.

Step 4.1: Move all the edges from ET with AET that corresponds to the value of Y
Sort the edges in AET in the increasing order of x_{min}

Step 4.2: Delete those edges from AET that has $y_{max} = y$
Sort the edges based on x_{min} value

Step 4.3: Update $y = y + 1$
Step 4.4: Fill pair of pixels b/w (y, x_{min}) of edges
Step 4.5: Update the ~~remaining~~ x_{min} of remaining edges as $x = x + 1/m$

Problems

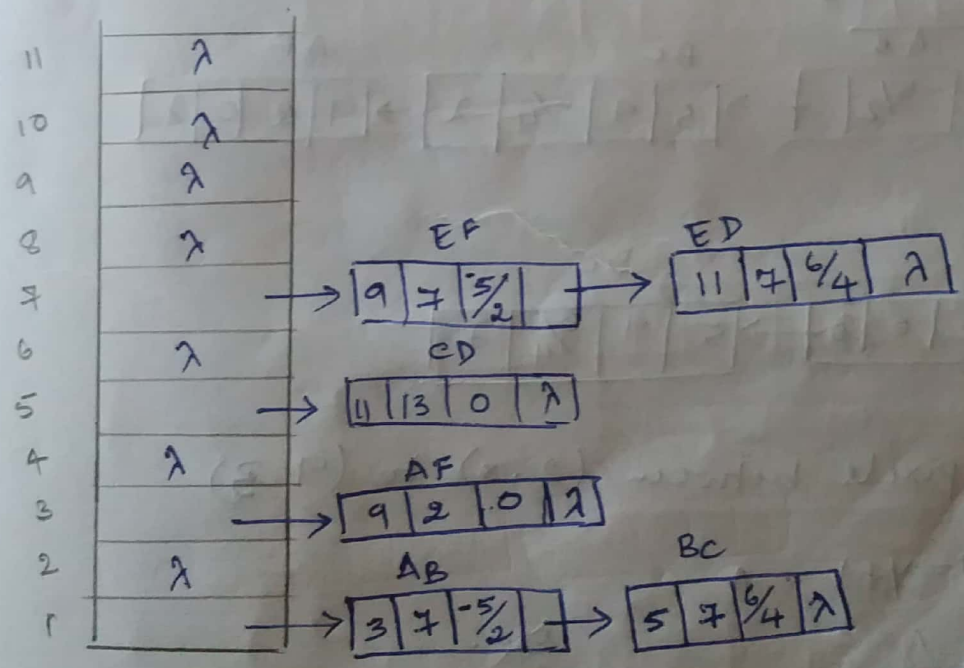


$\frac{11-7}{1-3}$

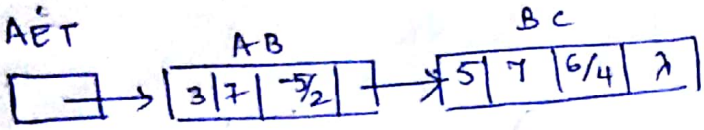
A(2,3)
B(7,1)
C(13,5)
D(13,11)
E(7,7)
F(2,9)

A(2,3)
B(7,1)
C(13,5)

D(13,11)
E(7,7)
F(2,9)

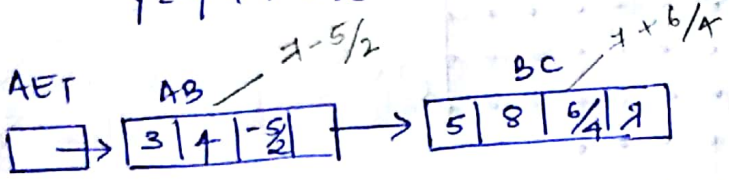


At scan line $y=1$

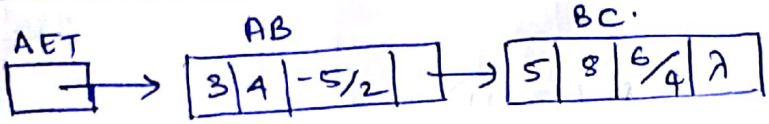


Fill pixels between (7, 1) to and (7, 1)

$y = y + 1 = 2$

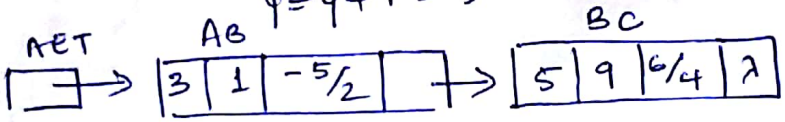


At scan line $y = 2$

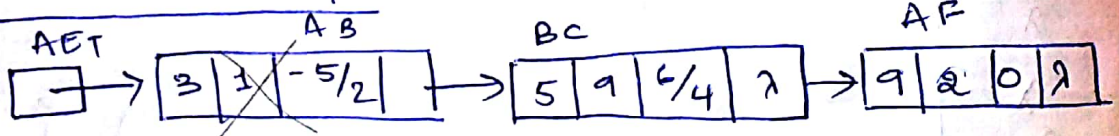


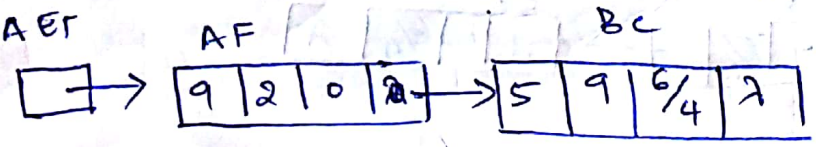
Fill pixels between (4, 2) to (8, 2)

$y = y + 1 = 3$



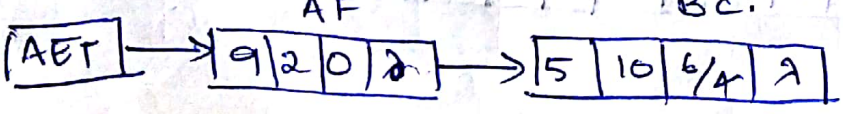
At scan line $y = 3$



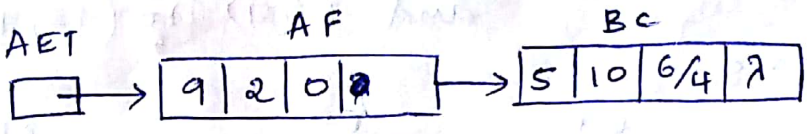


Fill pixels between (2, 3) to (9, 3)

$y = y + 1 = 4$

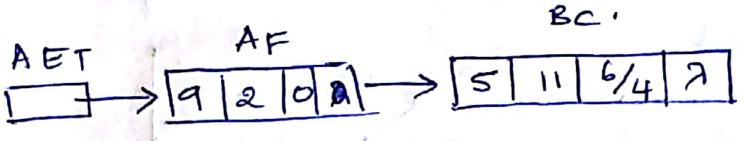


At scan line $y = 4$

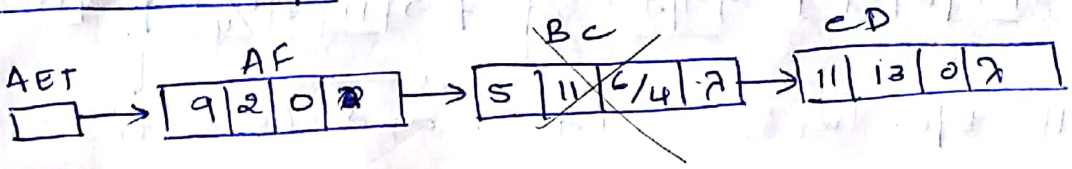


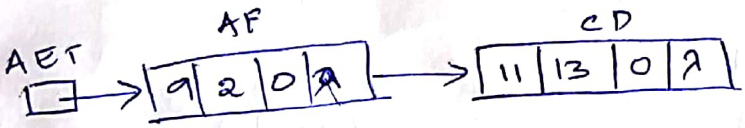
Fill pixels between (2,4) and (10,4)

$y = y + 1 = 5$.



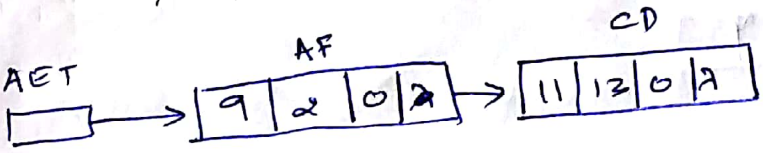
At scan line $y = 5$



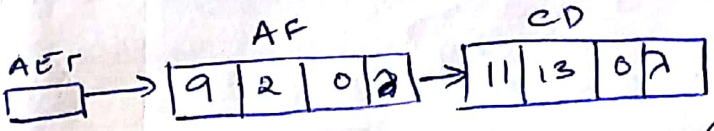


Fill pixels between (2,5) to (13,5)

$y = y + 1 = 6$.

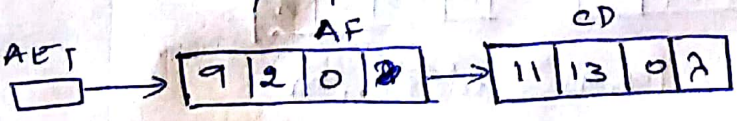


At scan line $y = 6$

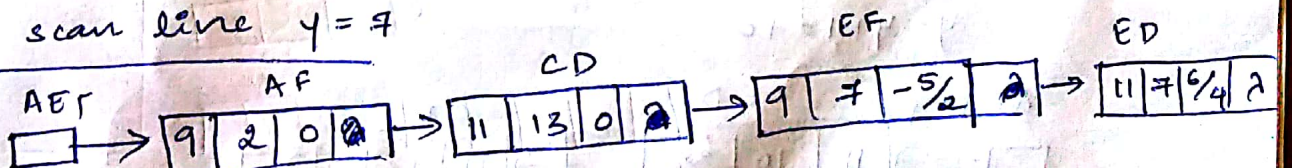


Fill pixels between (2,6) to (13,6)

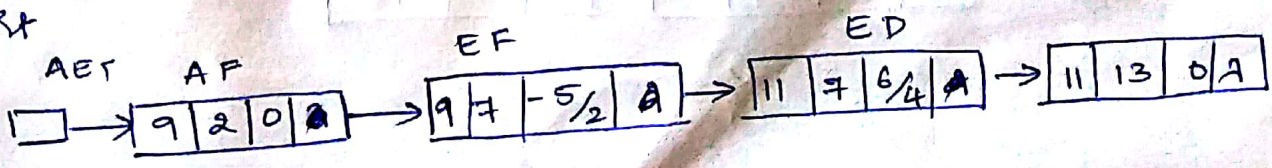
$y = y + 1 = 7$



At scan line $y = 7$

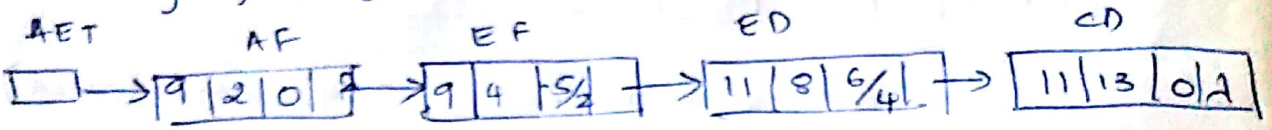


SORT

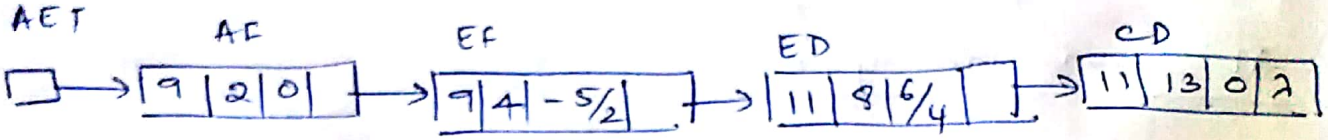


Fill pixels between (2, 7) to (9, 7)
and (7, 7) to (13, 7)

$y = y+1 = 8$.

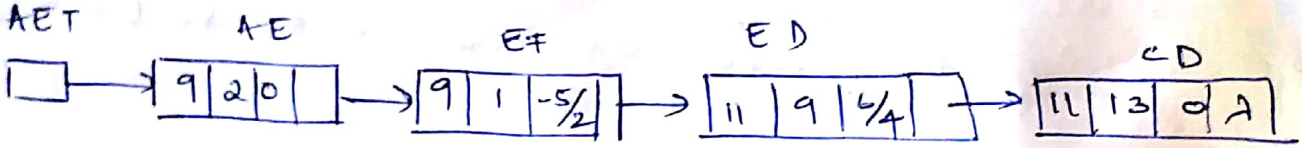


At scan line $y = 8$

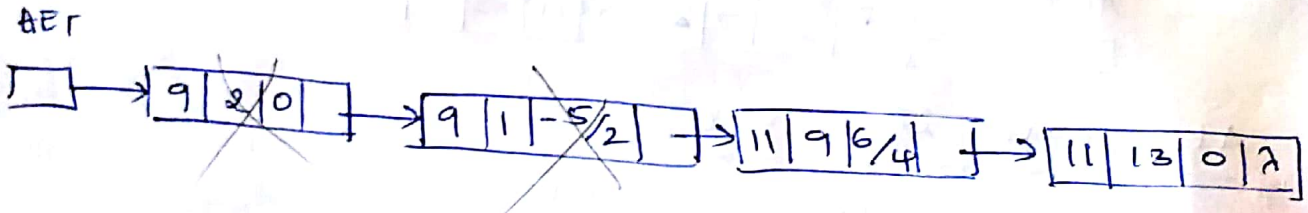


Fill b/w (2, 8) to (4, 8) and (8, 8) to (13, 8)

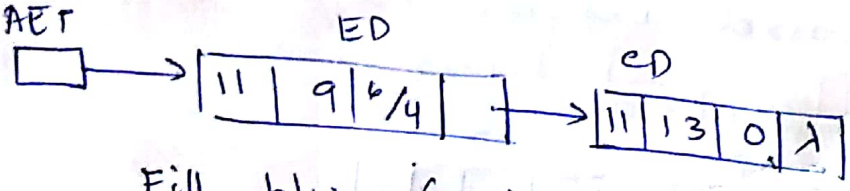
$y = y+1 = 9$



At scan line $y = 9$

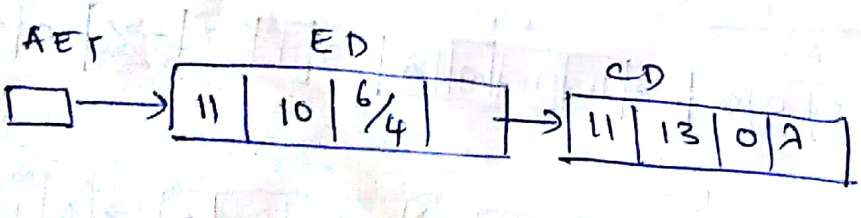


$y = y+1$

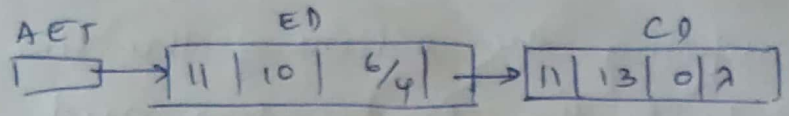


Fill b/w (9, 9) and (13, 9)

$y = y+1 = 10$

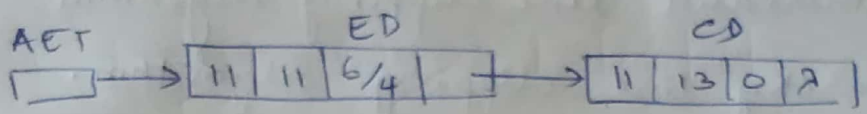


At scan line $y = 10$.

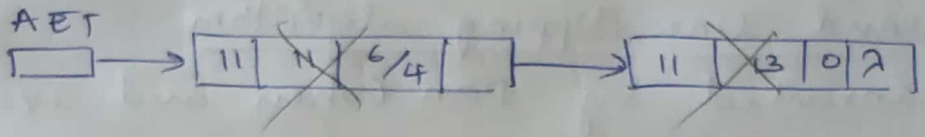


Fill b/w (10, 10) to (13, 10)

$y = y + 1 = 11$



At scan line $y = 11$

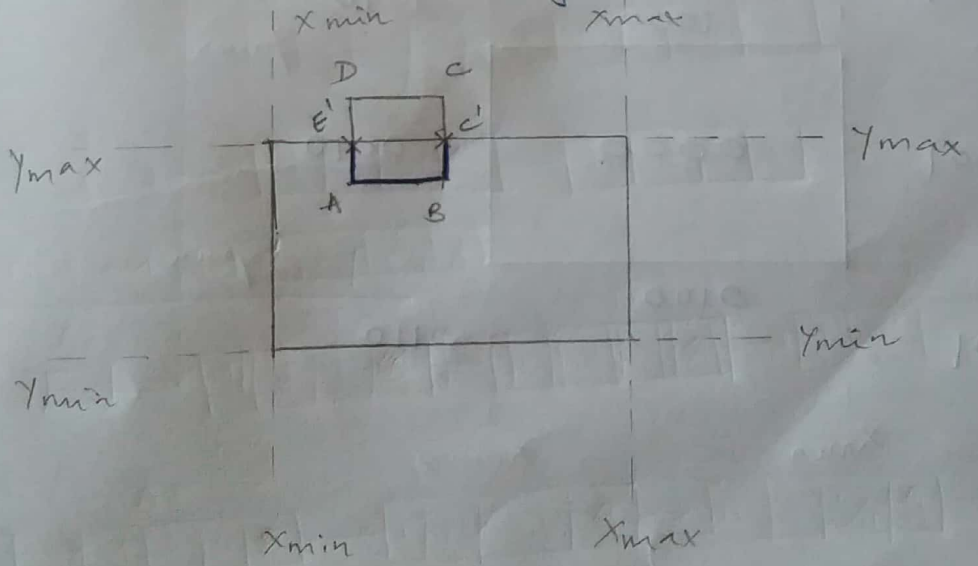


imp

Clipping

Clipping is a process of identifying the portions of picture lying inside or outside the specific area of interest and displaying those portions that lies inside the area of interest. called as clip window

- clip window is basically a rectangular region against which the object is clipped.



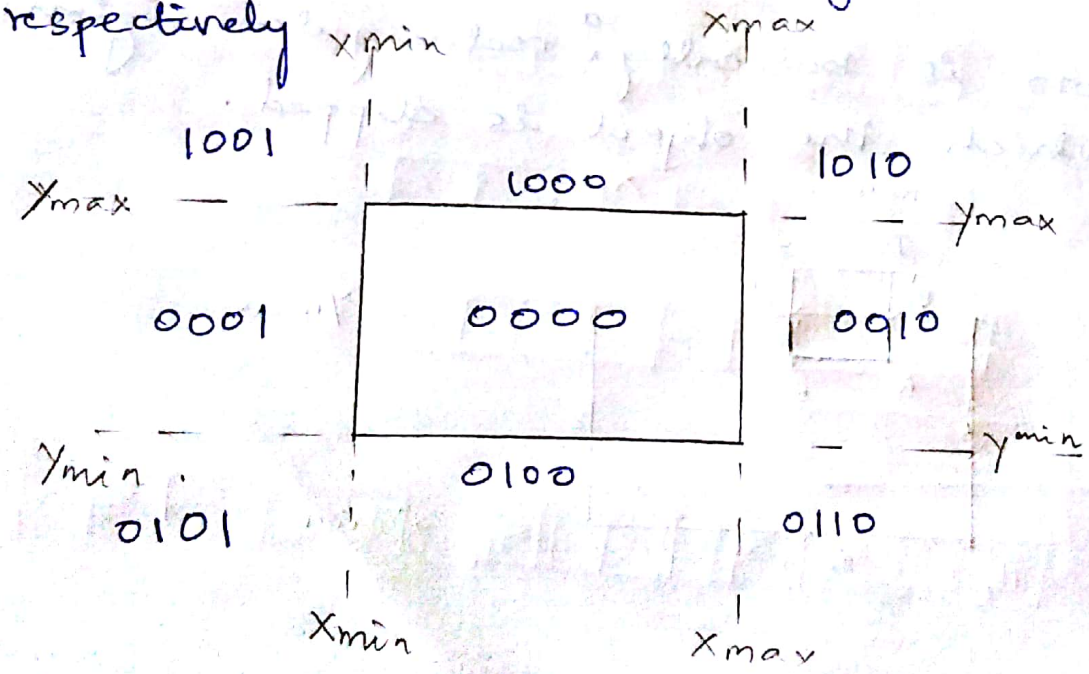
Line clipping

Process of identifying the portions of the line lying inside & outside the clip window and displaying those portions that lie inside the clip window

1) Cohen - Sutherland line clipping algorithm
2) Liang - Barsky line clipping algorithm.

Cohen-Sutherland line clipping algorithm

- They are invented by Dan Cohen and Ivan Sutherland
- This algorithm basically divides the clip window into 9 regions called as regions and for each region it assigns a four bit unique code called as region codes and region code is being assigned in the sequence TBRL where T bit refers to top B bit refers to bottom and R and L bit refers to right and left respectively



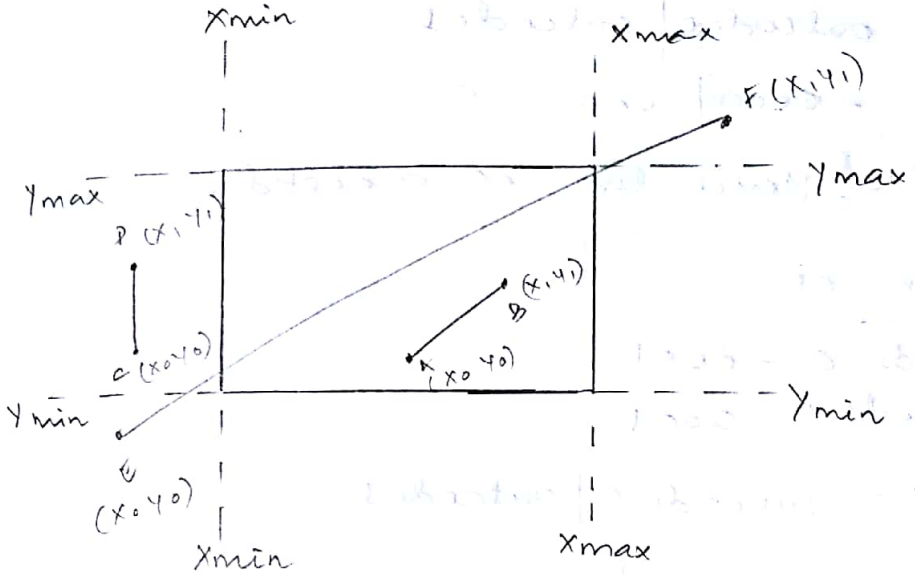
for bit T : if $Y > Y_{max}$, T=1

for bit B : if $Y < Y_{min}$ B=1

for bit R : if $X > X_{max}$, R=1

for bit L : if $X < X_{min}$, L=1

} using this we define a region code.



A(X0 Y0) = outcode 0 - 0000

B(X1, Y1) = outcode 1 - 0000

C(X0 Y0) = outcode 0 - 0001

D(X1,Y1) = outcode 1 = 0001

E(X0 Y0) = outcode 0 = 0101

F(X1,Y1) = outcode 1 = 1010

Three Basic tests

test1: if outcode 0 | outcode 1 == 0, ~~This~~ Line is completely inside hence accept it.

test2: if outcode 0 & outcode 1 != 0, Line is completely outside, hence reject it.

test 3: If not test 1 and test 2, Line is partially inside and outside, Hence it

Cohen Sutherland Clipping

For line AB

outcode 0 - 0000
outcode 1 - 0000

test 1: outcode 0 | outcode 1
= 0000 | 0000 = 0

⇒ Hence line is accepted.

For line CD

outcode 0 - 0001
outcode 1 - 0001

test 1: outcode 0 | outcode 1
= 0001 | 0001 = 0001

⇒ Test 1 is false

test 2: outcode 0 & outcode 1
= 0001 & 0001 = 0001 != 0

⇒ test 2 is true
Hence the line is rejected.

For line EF

outcode 0 - 0101
outcode 1 - 1010.

test 1: outcode 0 | outcode 1
= 0101 | 1010 = 1111

⇒ test 1 is false

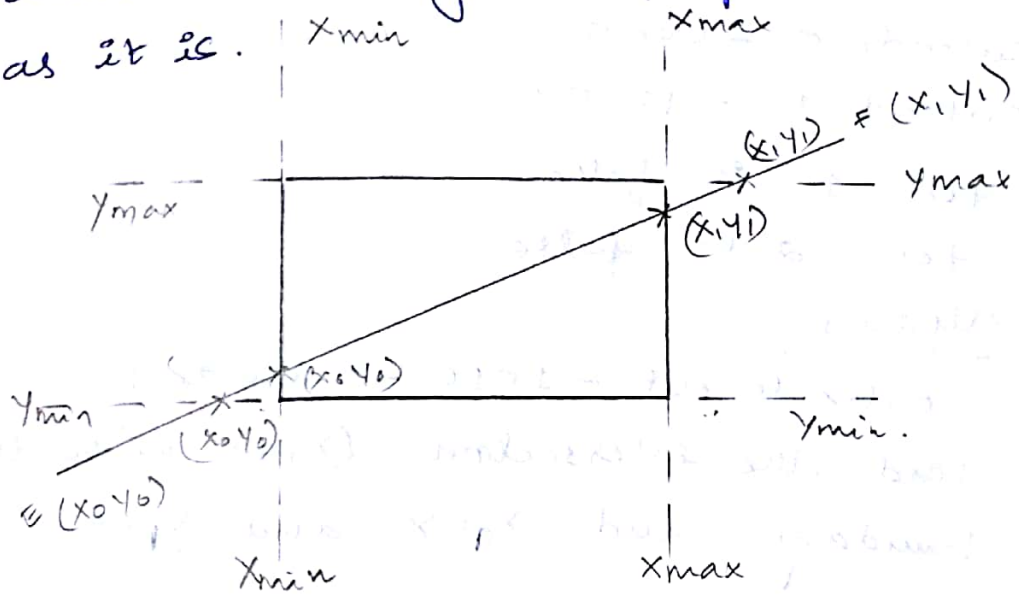
test 2: outcode 0 & outcode 1
0101 & 1010 = 0000

⇒ test 2 is false

test 3: Line needs clipping.

conditions for clipping

1. At a time only one end of the line could be clipped
2. At a time an end point of a line could be clipped only with respect to any one boundary.
3. When any end point is clipped the outcode of that end point changes and also its (x,y) coordinate changes keeping another endpoint as it is.



1) outcode 0 - 0101
outcode 1 - 1010

- test 1 is false
- test 2 is false.

clipping

Outcode out = outcode 0 = 0101

Find the intersection with bottom boundary and $x_0 = x$ and $y_0 = y$

outcode 0 now changes and becomes 0001

2) out code 0 - 0001
out code 1 - 1010.

test 1 is false
test 2 is false.

clipping

Outcode out = outcode 0 = 0001
Find the intersection. (X,Y) with left boundary and $x_0 = x$ and $y_0 = y$

Outcode 0 = 0000.

3) Outcode 0 - 0000
outcode 1 - 1010.

test 1 is false
test 2 is false

clipping

outcode out - 1010 - outcode 1
Find the intersection (X,Y) with top boundary and $x_0 = x$ and $y_0 = y$

outcode 1 = 0010

4) outcode 0 - 0000
outcode 1 - 0010

test 1 is false
test 2 is false

clipping

outcode out - outcode 1 - 0010
Find the intersection (X,Y) with right boundary and $x_1 = x$ and $y_1 = y$

outcode 1 = 0000

5) outcode 0 - 0000
outcode 1 = 0000

test 1 is true
Hence accept the clipped line.

Formula for finding the intersection

General form to find an intersection y & x. with respect to a line having endpoint $(x_0 y_0)$ & (x_1, y_1)

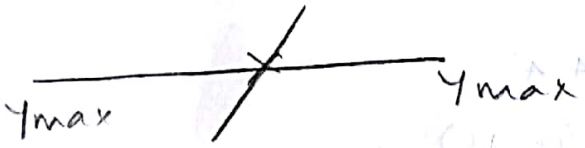
$$y = y_0 + m(x - x_0)$$

$$= y_0 + \frac{(y_1 - y_0)(x - x_0)}{(x_1 - x_0)}$$

$$x = x_0 + \frac{1}{m}(y - y_0)$$

$$= x_0 + \frac{(x_1 - x_0)(y - y_0)}{(y_1 - y_0)}$$

Intersection at top boundary.



$y = y_{max}$

$$x = x_0 + \frac{(x_1 - x_0)(y_{max} - y_0)}{(y_1 - y_0)}$$

Intersection at bottom boundary

$y = y_{min}$

$$x = x_0 + \frac{(x_1 - x_0)(y_{min} - y_0)}{(y_1 - y_0)}$$

Intersection at right boundary

$x = x_{max}$

$$y = y_0 + \frac{(y_1 - y_0)(x_{max} - x_0)}{(x_1 - x_0)}$$

Intersection at left boundary

$x = x_{min}$

$$y = y_0 + \frac{(y_1 - y_0)(x_{min} - x_0)}{(x_1 - x_0)}$$

Problem

Solve the problem using cohen Sutherland line clipping algorithm with a clip rectangle given with the minimum coordinates at (4,4) and maximum coordinates at (10,10) and line is given with respect to clip window having the coordinate end points at (6,3) and (12,11)

$(x_{min}, y_{min}) - (4,4)$

$(x_{max}, y_{max}) - (10,10)$

$x_0 = 6 \qquad y_0 = 3$

$x_1 = 12 \qquad y_1 = 11$

1]

outcode 0 = 0100

outcode 1 = 1010

test 1

outcode 0 | outcode 1 != 0

⇒ test 1 is false

test 2

outcode 0 & outcode 1 = 0

⇒ test 2 false

clipping

outcode out = outcode 0 = 0100

outcode out & top = 0100 & 1000 = 0

outcode out & bottom = 0100 & 0100 = 0100 != 0

clip against bottom

$y = y_{min} = 4$

$$x = x_0 + \frac{(x_1 - x_0)(y_{min} - y_0)}{(y_1 - y_0)}$$

$$= 6 + \frac{(12-6)(4-3)}{(11-3)} = 6 + \frac{(6)(1)}{8} = 6 + \frac{3}{4}$$

$$= \frac{54}{8} = 6.75$$

$x_0 = x = 6.75$

$y_0 = 4 = y$

outcode 0 = 0000

outcode 1 = 1010.

2) test 1:

outcode 0 | outcode 1 != 0

⇒ test 1 is false

test 2:

outcode 0 & outcode = 0

⇒ test 2 is false

clipping

outcode out = outcode 1 = 1010

outcode out & top = 1010 & 1000 = 1000 != 0.

Clip against top

$y = y_{max} = 10$

$$x = x_0 + \frac{(x_1 - x_0)(y_{max} - y_0)}{(x_1 - x_0)}$$

$$= 6.75 + \frac{(12 - 6.75)(10 - 4)}{(11 - 4)}$$

$$= 6.75 + \frac{(5.25)(6)}{7} = 11.25$$

$x_1 = x = 11.25$

$y_1 = y_{max} = y = 10.$

Outcode1 = 0010

Outcode 0 = 0000

3) test 1

Outcode 0 | Outcode 1 = 0010 != 0

⇒ test 1 is false

test 2

outcode 0 & outcode 1 = 0

⇒ test 2 is false

clipping

outcode out = outcode 1 = 0010

outcode out & top = 0010 & 1000 = 0

Outcode out & bottom = 0

outcode out & right = 0010 & 0010 != 0.

Clip against Right

$x = x_{max} = 10$

$$y = y_0 + \frac{(y_1 - y_0)(x_{max} - x_0)}{(x_1 - x_0)}$$

$$= 4 + \frac{(10-4)(10-6.75)}{(11.25-6.75)}$$

$$= 8.34$$

$x_1 = 10$

$y_1 = 8.34$

outcode 1 = 0 0 0 0

outcode 0 = 0 0 0 0

4) test 1

outcode 0 | outcode 1 = 0

⇒ test 1 is true

⇒ line is accepted.

$(x_0, y_0) = (6.25, 4)$ $(x_1, y_1) = (10, 8.34)$

Cohen - Sutherland Line clipping Algorithm

top = 8

bottom = 4

Right = 2

Left = 1

```
int compute code (int x, int y, int xmin, int ymin,
                  int xmax, int ymax)

{
   int code 0 = 0;
```

```
if (y > ymax)
    {
      code = code | top;
    }
if (y < ymin)
      {
        code = code | bottom;
      }
  if (x > xmax)
      {
        code = code | Right;
      }
  if (x < xmin)
      {
        code = code | Left;
      }
  return code;
}

void cohenSutherland (int x0, int y0, int x1, int y1, int
                      xmin, int ymin, int xmax, int ymax)
{
    outcode0 = computecode (x0, y0, xmin, ymin, xmax, ymax);
    outcode1 = computecode (x1, y1, xmin, ymin, xmax, ymax);
    do
      {
        if (outcode0 | outcode1 == 0)
            {
              accept = true;
              done = true;
            }
```

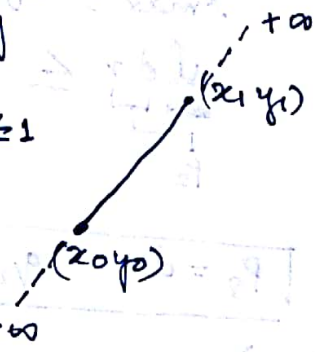
```
else if (outcode0 & outcode1 != 0)
    {
      accept = false;
      done = true;
    }

else
    {
      if (outcode 0)
        {
          outcodeout = outcode 0;
        }
      else
        {
          outcodeout = outcode 1;
        }
        if (outcodeout & top)
          {
            y = ymax ;
            x = x0 + (x1 - x0) * (ymax - y0) ;
                     ---------------------
                          (y1 - y0)
          }
        else if (outcode out & bottom)
          {
            y = ymin ;
            x = x0 + (x1 - x0) * (ymax - y0)
                     ---------------------  ;
                          (y1 - y0)
          }
        else if (outcodeout & Right)
          {
            x = xmax ;
            y = y0 + (y1 - y0) * (xmax - x0) ;
                     ---------------------
                          (x1 - x0)
          }
```

```
        else
          {
           x = xmin;
           y = y0 + (y1-y0)(xmin-x0) / (x1-x0);
          }

      if(outcode out == outcode 0)
          {
           x0 = x;
           y0 = y;
           outcode 0 = computecode(x0, y0, xmin, ymin,
                                   xmax, ymax);
          }
       else
          {
           x1 = x;
           y1 = y;
           outcode 1 = computecode(x1, y1, xmin, ymin,
                                   xmax, ymax);
          }
    }
  } while(done != true)

  if (accept == true)
     {
     drawline(x0, y0, x1, y1);
     }
}
```

Liang Barsky Line-clipping Algorithm

- It is based on a parametric equation for a line.
- A parametric equation for a line with endpoints (x_0, y_0) & (x_1, y_1) is given by

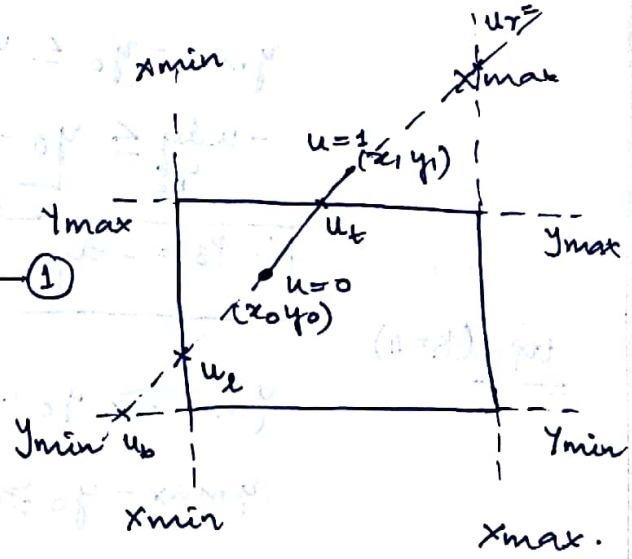
$$x = x_0 + u \cdot dx$$
$$y = y_0 + u \cdot dy \quad , \quad 0 \le u \le 1$$



- For an infinite line the parametric equation is

$$x = x_0 + u dx \qquad -\infty \le u \le +\infty$$
$$y = y_0 + u dy$$

$$x_{min} \le x_0 + u.dx \le x_{max}$$
$$y_{min} \le y_0 + u dy \le y_{max} \quad \Big\} \text{—— (1)}$$



$$\boxed{\begin{aligned} u_1 &= \max(0, u_l, u_b) \\ u_2 &= \min(1, u_t, u_r) \end{aligned}}$$

Consider the equation (1) and write 4 inequalities $k = 1, 2, 3, 4$, were the inequalities represent left, right, bottom and top boundary and write each inequality in the form of,

$$\boxed{p_k \cdot u \le q_k}$$

left $(k=1)$ $\quad x_{min} \le x_0 + u dx.$

$$\Rightarrow x_{min} - x_0 \le u \cdot dx$$
$$-u \cdot dx \le x_0 - x_{min} \quad \text{—— (1)}$$
$$\underbrace{\quad}_{p_1} \qquad \underbrace{\quad}_{q_1}$$

$p_1 = -dx$, $q_1 = x_0 - x_{min}$.

Right ($k=2$)

$$x_{max} \geq x_0 + u dx.$$

$$u dx \leq x_{max} - x_0 \quad \text{—— (ii)}$$

(p_2 : $u dx$, q_2 : $x_{max} - x_0$)

$$\boxed{p_2 = dx \quad q_2 = x_{max} - x_0}$$

bottom ($k=3$)

$$y_{min} \leq y_0 + u dy.$$

$$y_{min} - y_0 \leq u dy.$$

$$-u dy \leq y_0 - y_{min} \quad \text{—— (iii)}$$

(p_3 : $-u dy$, q_3 : $y_0 - y_{min}$)

$$\boxed{\therefore p_3 = -dy \quad , \quad q_3 = y_0 - y_{min}.}$$

top ($k=4$)

$$y_{max} \geq y_0 + u dy.$$

$$y_{max} - y_0 \geq u dy$$

$$u dy \leq y_{max} - y_0. \quad \text{—— (iv)}$$

$$\boxed{p_4 = dy \quad q_4 = y_{max} - y_0}$$

Algorithm

Step1 : Derive 4 inequalities for p_1 p_2 p_3 p_4 and q_1 , q_2 , q_3 , q_4

Step2 : For any k $p_k = 0$ and $q_k \leq 0$, then reject the line since it is outside and stop. else step 3.

Step3 : For any k if $p_k < 0$. calculate $r_k = \frac{q_k}{p_k}$

and set $u_1 = \max(0, \text{calculated } r_k \text{ values})$

step 4: For any k if $P_k > 0$, calculate $r_k = \frac{q_k}{P_k}$

and set $u_2 = \min(1, \text{calculated } r_k \text{ values})$

step 5: If $u_1 > u_2$, line is outside and reject it and stop else go to step 6.

step 6: $x = x_0$, $y = y_0$

$x_0 = x + u_1 dx$

$y_0 = y + u_1 dy$

$x_1 = x + u_2 dx$.

$y_1 = y + u_2 dy$

Problems

9] Solve the given problem by the method of Liang Barksy line clipping algorithm where the clip rectangle given by the minimum coordinates (4,4) and maximum coordinates (10,10) with the line given with an endpoint (6,3) and (12,11)

$x_{min} = 4$ $y_{min} = 4$

$x_{max} = 10$ $y_{max} = 10$

$x_0 = 6$ $y_0 = 3$

$x_1 = 12$ $y_1 = 11$

$dx = 6$ $dy = 8$

$P1 = -dx = -6.$ $q_1 = x_0 - x_{min} = 6 - 4 = 2$

$P2 = dx = 6$ $q_2 = x_{max} - x_0 = 10 - 6 = 4$

$P3 = -dy = -8$ $q_3 = y_0 - y_{min} = 3 - 4 = -1$

$P_4 = dy = 8$ $q_4 = y_{max} - y_0 = 10 - 3 = 7$

For $P_k < 0$, $q_1 = \frac{q_1}{P_1} = \frac{2}{-6} = -1/3 = -0.33$

$$q_3 = \frac{q_3}{P_3} = \frac{-1}{-8} = 0.125$$

$u_1 = \text{amax}\,(0, -0.33, 0.125)$

$\therefore u_1 = 0.125$

For $P_k > 0$

$$q_2 = \frac{q_2}{P_2} = \frac{4}{6} = 0.667$$

$$q_4 = \frac{q_4}{P_4} = \frac{7}{8} = 0.875$$

$u_2 = \min(1, 0.67, 0.875)$

$\therefore u_2 = 0.67$

$x = 6, \quad y = 3$

$x_0 = x + u_1 dx = 6 + (0.125)(6) = 6.75$

$y_0 = y + u_1 dy = 3 + (0.125)(8) = 4$

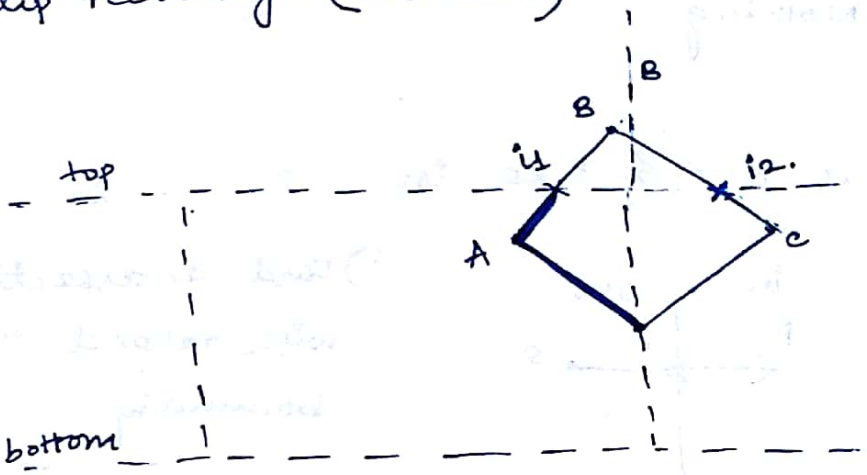
$x_1 = x + u_2 dx = 6 + (0.67)(6) \simeq 10.$

$y_1 = y + u_2 dy = 3 + (0.67)(8) = 8.36.$

Polygon clipping

- clipping a given polygon against a clip window which represents a rectangle.
- we identify the portion of a polygon that is outside and inside with respect to the clip rectangle and display those portions which are only inside the clip rectangle (window)

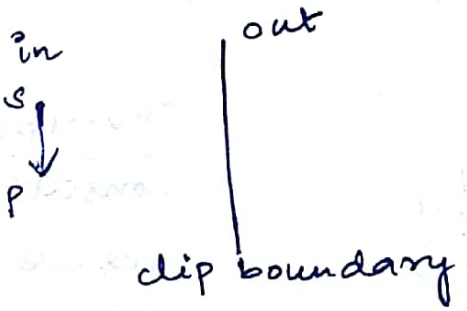
Ex



Sutherland - Hodgeman polygon clipping Algorithm

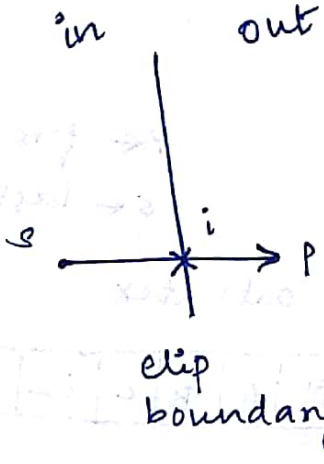
- It is associated with 4 test.

test 1: s is inside, p is inside



output: p

test 2: s is inside & p is outside.

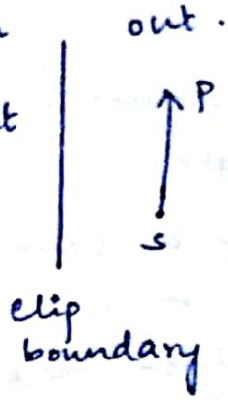


i) Find intersection i with respect to clip boundary

output: i

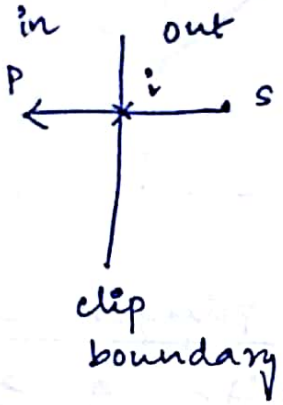
test 3:

s is out & p is out



output : Nothing.

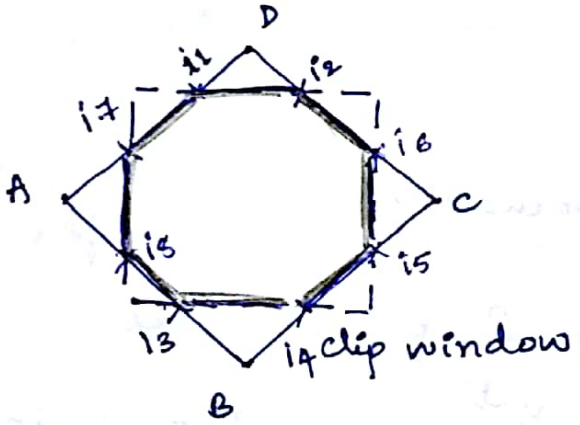
test 4: s is out & p is in



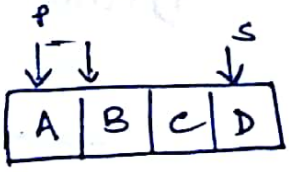
i) Find intersection i with respect to clip boundary

output : add i
add p

Problems



invertex → An array consisting of a list of vertices either in clockwise or anti-clockwise direction

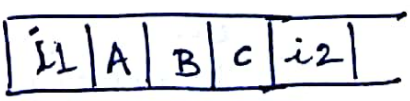


invertex

P ← first position of array
s ← last position of array

consider top boundary

outvertex



1) S = D, out
P = A, in

Find intersection i1

Output: i1
A

Outvertex: An array that consist of vertices considered after clipping
It gets updated in every iteration

2) S = P = A, in
P = B, in

output: B

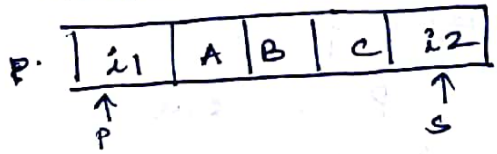
3) S = P = B, in
P = C, in.

output: C

4) S = P = C, in
P = D, out
Find intersection i2
Output: i2.

[Before going to next boundary we need to update invertex. Invertex gets all the elements of outvertex. outvertex for next boundary starts from null]

invertex

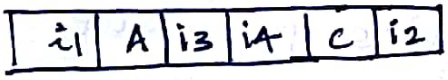


consider bottom boundary

1) P = i1, in
S = i2, in

Output: i1

outvertex



2) S = P = i1, in
P = A, in

Output: A

3) S=P=A, inside

P= B, out

Find the intersection is

output: i3

4) S=P=B, out

P=C, in

Find intersection i4, output: i4

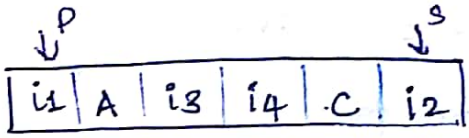
C

5) S=P=C, in

P=i2, in

output: i2.

updated inverter



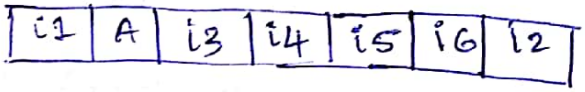
Consider right boundary

1) P=i1, in

S=i2, in

output i1

outverter



2) S=P=i1, in

P=A, in

output: A

3) S=P=A, in

P=i3, in

output: i3

4) S=P=i3, in

P=i4, in

output: i4

5) S=P=i4, in

P=C, out

Find intersection is

output: i5

6) S=P=C, out

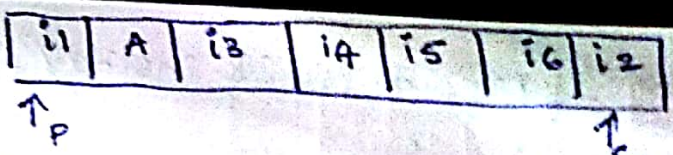
P=i2, in

Find intersection i6

Output: i6

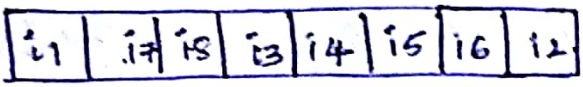
i2

update invertex



consider the left boundary

outvertex



1) p = i1, in
s = i2, in
output: i1

2) s = p = i1, in
p = A, out
Find intersection i7
output: i7

3) s = p = A, out
p = i3, in
Find intersection i8.
output i8
i3

4) s = p = i3, in
p = i4, in
output: i4

5) s = p = i4, in
p = i5, in
output: i5

6) s = p = i5, in
p = i6, in
output i6

7) s = p = i6, in
p = i2, in
output: i2.

Algorithm

```
Sutherland-Hodgeman (clipboundary, invertex[],
                                    outvertex[])
{
  outvertex[] = NULL;
  inlength = length (invertex)
    s = invertex [inlength-1];
    for (j=0; j < inlength ; j++)
      {
        p = invertex [j];
```

```
if (inside(s) && inside(p))
    {
    add(p, outvertex);
    }
if (inside(s) && outside(p))
    {
    i = intersection(clipboundary);
    add(i, outvertex);
    }
if (outside(s) && inside(p))
    {
    i = intersection(clipboundary);
    add(i, outvertex);
    add(p, outvertex);
    }
s = p;
}
outlength = length(outvertex);
for (j=0; j < outlength; j++)
    {
    invertex[j] = outvertex[j];
    }
}

void polygonclip(invertex[], outvertex[])
    {
    invertex = list of vertical traversed in
               anticlockwise direction;
```

```
Sutherland_Hodgeman(top, invertex[], outvertex[]);
Sutherland_Hodgeman(bottom, invertex[], outvertex[]);
Sutherland_Hodgeman(right, invertex[], outvertex[]);
Sutherland_Hodgeman(left, invertex[], outvertex[]);
}
```

UNIT - II

2D Transformation

- A 2D transformation is a process of changing the 2D graphics image that is projected over 2D plane.

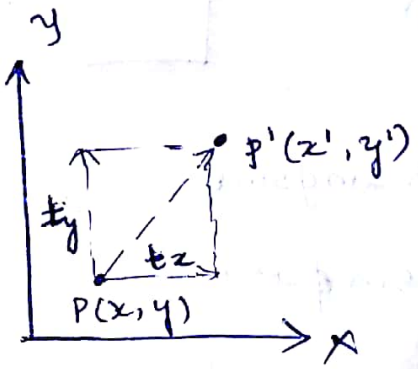
3 types of transformation

1. Translation
2. Rotation
3. Scaling.

1] Translation

Translation is a process of changing the position of an object from one point to another point along the x and y axis over the 2D plane.

Derivation



$x' = x + t_x$

$y' = x + t_y$